UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA INFORMÁTICA

PROYECTO FIN DE CARRERA

Trusted Authetication Protocol for Self-Organizing Networks

Autor: Elisa Pintado Guijarro

Indice

1 INTRODUCCIÓN

Las redes P2P están basadas en una arquitectura distribuida que cubre un conjunto de tipos de redes. La propiedad común compartida por casi todas las redes P2P es la ausencia de un control centralizado. Este tipo de redes son la antítesis del modelo cliente-servidor tradicional. Se han hecho muy populares para compartir archivos.

En este proyecto se demuestra como algunas propiedades de la especificación TCG (*Trusted Computing Group*) pueden ser utilizadas para mejorar la seguridad en redes P2P.

Los pasos realizados para elaborar el proyecto han sido:

- En primer lugar se ha llevado a cabo un estudio del estado del arte en el cual se han investigado las distintas soluciones que ofrece *Trusted Computing* y su aplicación en redes P2P.

- Aprendizaje y uso de las herramientas *TPM Tools*, conjunto de programas y librerías.

- A continuación se ha realizado el diseño del protocolo basado en una publicación del grupo *IAIK SCOS* el cual explica una solución de forma teórica.

- También se ha llevado a cabo una familiarización inicial de los algoritmos criptográficos y mecanismos como cifrado simétrico y asimétrico, generación de número aleatorios de forma segura, funciones resumen *(hash)* y firmas digitales. Utilizando estos mecanismos, se ha diseñado un protocolo seguro utilizando distintos métodos del área de *Trusted Computing*. Un protocolo utiliza el concepto de *PrivacyCA* para autenticación y protección de la privacidad mientras que el otro protocolo se basa únicamente en la utilización del módulo TPM *(Trusted Platform Module).*

Este último diseño es el que más encaja con la filosofía P2P dado que la ausencia de la entidad *PrivacyCA* deja a todos los nodos de la red como iguales. De esta forma todos los nodos trabajan de forma distribuida.

2 ESTADO DEL ARTE

Actualmente no existe una solución de autenticación segura con *Trusted Computing*, aunque si que se han redactado artículos aportando una solución teórica. Los problemas que existen en las redes P2P son una amenaza a la seguridad bien conocido. No ha habido muchas investigaciones para identificar y tratar de resolver algunos de estos temas. En teoría, todos los nodos que pertenecen a una red P2P proporcionan la misma cantidad de recursos al sistema. Pero esto no siempre es cierto, algunos nodos tratan de perturbar la ejecución del sistema.

Dependiendo del tipo de red P2P varios problemas pueden surgir. Por ejemplo, pueden existir nodos que tratan de infectar una red con archivos maliciosos, otros consumen mucho ancho de banda y comparten escasos recursos con el resto de nodos. Como se describe en **[4]**, los problemas en las capas de P2P se pueden separar en dos dominios: relativo al sistema y relativo a la aplicación específica.

Stefan Kraxberger et al. **[4]** ofrece una solución basada en el uso de la información y el conjunto de claves encerradas dentro del TPM de Trusted Computing. Este TPM es utilizado como un módulo confiable para el almacenamiento de certificados y para autenticar la información.

En nuestro trabajo se describen los distintos enfoques realizados para resolver los problemas conocidos de autenticación en redes P2P. Nuestra solución se basa en las directrices establecidas por el grupo IAIK en **[4]**, donde podemos encontrar una explicación detallada sobre la creación de identidades de confianza para redes P2P.

2.1 Plataformas de Trusted Computing.

El Trusted Computing Group (TCG) **[2]** es una organización integrada por fabricantes de software, hardware, sistemas y redes e infraestructuras, que nació en el 2003 con el fin de desarrollar y promover estándares de Trusted Computing.

Este grupo ha definido un conjunto de especificaciones para tender a convertir las arquitecturas actuales de los equipos en plataformas de confianza. Con la intención de proporcionar medios hardware que ayuden en la confiabilidad el grupo ha especificado el Trusted Platform Module (TPM) **[8]**.

El TPM es un hardware con operaciones criptográficas que contiene un conjunto de herramientas que proporciona cifrado de clave pública, generación de claves, hash y generación de números aleatorios. De esta manera, los equipos tienen una capacidad que va más allá de la utilización de tarjetas inteligentes o tokens, asegurando que sólo los usuarios autorizados y las máquinas autorizadas van a estar en la red.

Además, el TPM proporciona un par de claves de cifrado que se integran de forma permanente dentro del hardware. Este par de claves es conocido como Endorsement Key (EK) y es creada generalmente en el momento de fabricación. La parte privada de la clave EK nunca sale de un TPM, de esta manera la parte pública de la clave EK se utiliza para reconocer el TPM original.

Otra función muy importante de la clave EK consiste en el uso para firmar datos y de eta manera poder averiguar si los datos son íntegros. Para realizar la firma de una parte de datos se utiliza la clave privada para cifrar un pequeño trozo de la información. Una vez firmado, esta firma

puede ser verificada mediante el uso de la clave pública correspondiente y así poder descifrar ese mismo elemento de datos. Si se puede descifrar con la clave pública, entonces debe haber sido cifrado con la clave privada correspondiente. Mientras que la clave privada del TPM sea mantenida en secreto, podrá utilizarse como una firma digital de confianza.

El TCG ofrece una infraestructura de software conocida como la pila de software del TCG (TSS) **[9]** mediante el cual las aplicaciones pueden acceder a las funciones de Trusted Computing y de esta manera utilizar servicios de confianza.

Para evitar la manipulación de la información correspondiente a una plataforma, el TPM almacena la información en forma de registros de configuración de la plataforma (PCR). Un informe sobre el estado de un PCR refleja estado exacto de un sistema, por lo que su autenticidad e integridad debe ser preservada. Con este informe, una máquina externa es capaz de informarse sobre la confiabilidad del sistema. Este concepto sobre una plataforma con Trusted Computing se conoce como autenticación a distancia (Remote Attestation). Con este fin, el TPM actúa como un elemento raíz de confianza para la presentación de informes (Core Root-Of-Trust for Reporting, CRTR).

Bajo petición, se lleva a cabo la operación del TPM llamada TPM_quote que firma el estado del PCR con una clave habilitada para firmar por el TPM. En una autenticación a distancia es importante saber que se está produciendo una comunicación con una plataforma TPM válida. Con este fin la clave EK puede ser utilizada, pero esta clave socavaría la privacidad del propietario, por lo que una clave específica es creada por el TPM para llevar a cabo la autenticación llamada Attestation Identity Key (AIK). Este clave AIK está formado por un par de claves de infraestructura pública.

2.2 PrivacyCA

En una autenticación a distancia con TC se requiere que una máquina envíe una descripción detallada del estado de su sistema, esto puede ser muy susceptible a ataques. Una solución lógica sería utilizar un servidor privado llamado PrivacyCA como una entidad de confianza que protege la privacidad de los usuarios. Más información acerca del funcionamiento de un PrivacyCA se detalla en **[6]**.

Uno de los objetivos en una autenticación con TC es permitir a la máquina que desea entrar en el sistema determinar que un TPM ha firmado un determinado mensaje pero sin saber que TPM lo ha firmado, es decir que el TPM sea anónimo para otro usuario.

La especificación del TCG **[10]** define el PrivacyCA como una entidad que se emplea para poder crear credenciales AIK que avalen la fiabilidad de una plataforma sin revelar los valores EK únicos de una máquina a otra máquina que desea autenticarse. Un TPM crea una clave pública AIK y las registra en un PrivacyCA, el cual podrá a continuación distribuir un certificado que certifique el AIK. Para registrar una clave AIK el TPM debe probar que esa clave esta ligada al TPM de forma exclusiva. Esto es logrado cuando el TPM descifra la credencial AIK con su clave EK privada. Unicamente el TPM propietario de la clave privada EK será capaz de realizar el descifrado.

2.3 Plataforma SEPP

La plataforma SEPP para P2P seguro **[7]** es un proyecto que implementa una infraestructura para securizar redes P2P utilizando Java. Otras plataformas existentes de P2P no consideran la seguridad a nivel de diseño, si no que se limitan a implementar algunos mecanismos de seguridad. En nuestro trabajo hemos implementado nuevas funcionalidades basadas en la autenticación utilizando Trusted Computing. Esta plataforma SEPP se utiliza como una herramienta para la obtención de una red segura P2P donde podemos desarrollar nuestra solución.

En esta plataforma se ha llevado a cabo el estudio de las ventajas y desventajas de utilizar distintos mecanismos de autenticación a través de los resultados obtenidos y a través de controlar el tiempo de ejecución de cada tipo de autenticación

3 Configuración de un TPM

En nuestra solución asumimos que las máquinas usadas utilizan el módulo TPM y están habilitadas para su uso. Esta sección describe como poder utilizar un TPM. A continuación se indica los requisitos del sistema y que paquetes de Java han sido utilizados.

3.1 Requisitos del sistema

El hardware TPM utilizado es Intel Infineon Trusted Platform Module, version 1.2 **[11]**.

En primer lugar, es necesario habilitar el TPM en la BIOS. Si una máquina no posee TPM es posible obtener un emulador de TPM **[12]**. El emulador TPM es un paquete para Linux que provee las funcionalidades de un TPM, estando implementadas mediante software.

El sistema operativo utilizado en nuestra solución ha sido Windows 7. El objetivo de este trabajo era desarrollar nuestra solución dando soporte para este sistema operativo, ofreciendo al usuario una alternativa respecto a Linux.

Para la implementación se ha utilizado un entorno Sun Java (tm) para la version 5 en adelante. Versiones anteriores de Java (tm) no tienen la funcionalidad criptográfica requerida. El entorno utilizado es Eclipse. Nuestro software ha sido implementado a partir de otras plataformas ya existentes. Los paquetes utilizados son.

- jTSS
- jUnit
- jTPMTools
- TCcert Tool
- PrivacyCA

3.2 Habilitar el TPM

Para utilizar un módulo hardware TPM el primer paso es activarlo en la BIOS. Para mayor información consultar el manual del equipo. Una instalación manual de JTSS para Linux y Windows se describe en **[13]**. Una vez habilitado el TPM, hay que establecer que eres el dueño de ese TPM, cuando esto se realiza, el storage root key (SRK) es almacenado en el sistema (sin la parte de la clave privada).

Para definir que eres el propietario o dueño del TPM se utiliza jTpm Tools. Para ello se utiliza el comando "take_ownership", existe una descripción detallada en **[14]**. Si el TPM ha sido obtenido por otro propietario anteriormente se puede utilizar el comando "clear_ownership" para borrar todos los datos previos. Estos comandos deben ser ejecutados con permisos de administrador.

Un TPM se inicia utilizando una contraseña de propietario , una contraseña SRK, una contraseña AIK y un identificador de etiqueta AIK. En Java esto se consigue de la siguiente manera.

• Owner password

```
ownerSecret = TcBlobData.newString("mypassword", appendNullTerm, encoding);
```

• AIK password (obligatorio)

```
aikSecret = TcBlobData.newString("mypassword", appendNullTerm, encoding);
```

• SRK password

```
srkSecret = TcBlobData.newByteArray(TcTssConstants.TSS WELL KNOWN
SECRET);
srkSecretMode = TcTssConstants.TSS SECRET MODE SHA1;
```

• AIK id label (obligatorio)

```
idLabel = TcBlobData.newString("myidlabel", false);
```

3.3 Otras herramientas

IAIK/OpenTC TCcert **[15]** es una herramienta software que permite la creación de tipos especiales de certificados, como especifica el Trusted Computing Group.

IAIK/OpenTC PrivacyCA **[16]** es un paquete que ofrece implementaciones básicas de un servidor PrivacyCa.

4 IMPLEMENTACIÓN

Nuestra aplicación se ha implementado basándonos en la plataforma *Trusted Computing for the Java™ platform* **[1]**, donde hemos modificado estructuras y creado nuevos métodos para lograr nuestro cometido. Esta solución ha sido integrada en la aplicación Sepp, donde hemos podido probar nuestro código en una red P2P utilizando como ejemplo un cliente chat.

Se ha modificado la aplicación Sepp **[7]** y además le hemos añadido funcionalidad. Estas Características incluyen dos tipos diferentes de autenticación que la infraestructura no soportaba:

- Autenticación fiable con la entidad *PrivacyCA*.
- Autenticación fiable sin la entidad *PrivacyCA*.

La primera utiliza una entidad *PrivacyCA* como un tercero de confianza. El segundo tipo soporta una autenticación segura sin la intervención de esta entidad *PrivacyCA*, utilizando sólo los módulos TPM de cada nodo, el cual es utilizado como *Root-of-Trust*.

4.1 Protocolo Seguro de Autenticación

El Protocolo Seguro de Autenticación, TAP, por sus siglas en inglés (Trusted Authentication Protocol), es utilizado en los dos nuevos tipos de autenticación. TAP está basado en el protocolo de Needham-Schroeder PK. Este protocolo añade números aleatorios en el proceso de autenticación explicado en el capítulo 10 del libro *Handbook of Applied Cryptography* **[3]**.

En la figura que se muestra a continuación se puede ver como es el intercambio de mensajes en el proceso de autenticación, para los dos tipos de autenticación (autenticación con y sin la entidad *PrivacyCA)*.

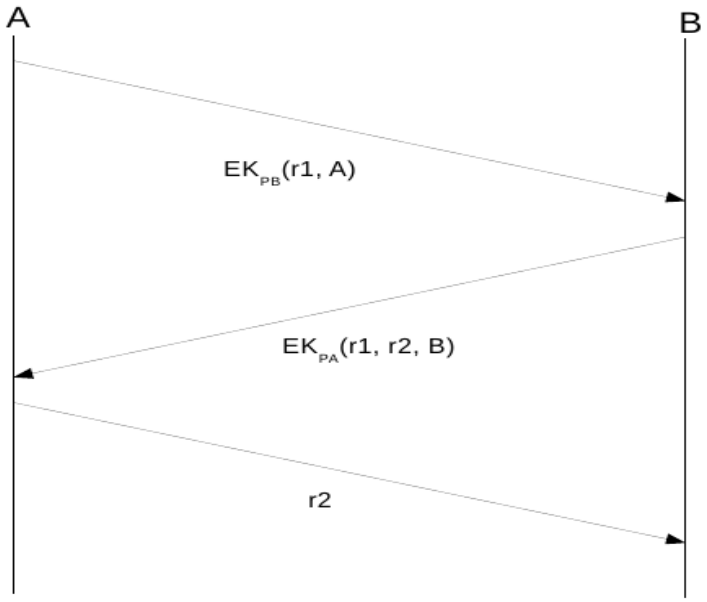


Figura 1: Trusted Authentication Protocol (TAP)

A: Nodo A
B: Nodo B
EK_{PA}: Clave EK pública de A
EK_{PB}: Clave EK pública de B
R1: Número aleatorio 1
R2: Número aleatorio 2

En el primer envío, el nodo A envía al nodo B un mensaje con su identificación y un número aleatorio, R1, que es generado por este nodo, los datos son cifrados con la clave EK pública de B. Después, B recibe el blob con los datos cifrados y los descifra con su privada, así obtiene el número aleatorio R1. En el siguiente paso, B envía a A su identificación, R1 y genera el número aleatorio R2, todos estos datos son cifrados en este caso con la clave pública de A. Finalmente, A recibe de la misma manera un blob cifrado con R2 y lo descifra para volver a enviarse a B, este mensaje no va

cifrado. Si el número que el nodo generó es igual al número que recibe de vuelta es igual, entonces el proceso será satisfactorio.

La siguiente sección explica con más detalle la implementación del protocolo TAP en los dos tipos de autenticación.

4.2 Autenticación segura con la entidad PrivacyCA

Este tipo de autenticación se basa en la solución expuesta en el artículo *Trusted Identity Management for P2P Networks* **[4]**, donde se explica cómo resolver los problemas de autenticación en redes P2P.

Se ha implementado una nueva clase llamada *TPMmanager* la cual maneja la comunicación con el módulo TPM. Con el fin de completar nuestra solución fue imprescindible modificar varias de las librerías de la plataforma *Trusted Java Framework* **[1]**para utilizarlas en nuestro programa, estos cambios se detallarán más adelante.

El proceso comienza cuando un usuario se quiere conectar a la red, este usuario se llamará *attestant.* Este *attestant* tiene asociada un certificado AIK que no contiene ninguna información que pueda relacionarse con ese nodo. Para impedir los posibles ataques, la entidad *PrivacyCA* tiene que chequear que un nodo con TPM o clave EK sólo puede utilizar un certificado AIK.
El flujo que sigue está basado en el protocolo AIK descrito en **[4]**.

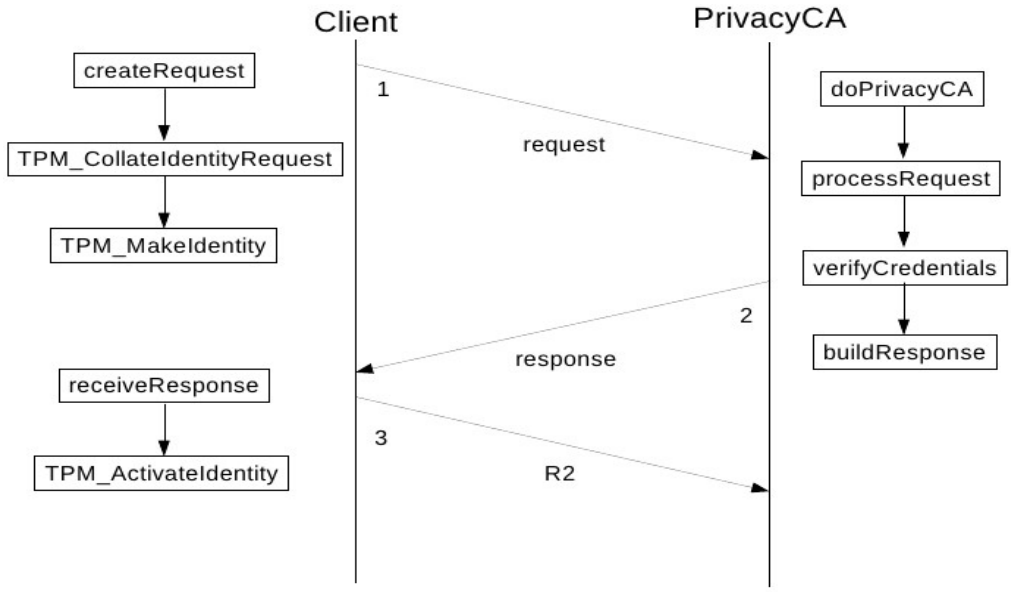


Figura 2: Trusted Authentication Protocol con PrivacyCA

1. El cliente prepara la solicitud para enviársela al *PrivacyCA*:
   - En el primer paso el cliente hace una llamada al método createRequest que invoca a la función TSS Tspi_TPM_CollateIdentityRequest para crear un nuevo AIK.
   - El TSS invoca a la función TPM_MakeIdentity para crear el nuevo AIK que es una pareja de claves RSA.
   - El TPM devuelve una estructura que contiene la clave AIK pública firmada con la clave AIK privada. A continuación el TSS genera una solicitud donde se almacenan los datos

en una estructura llamada *identityProof* que contiene el blob firmado por el TPM, el número aleatorio R1 y la clave EK y PE.
- Estos datos se cifran con la clave pública del *PrivacyCA.*
- El cliente envía la solicitud al *PrivacyCA.*

2. El *PrivacyCA* prepara la solicitud para enviársela al cliente:
   - El *PrivacyCA* hace una llamada al método processRequest para descifrar los datos y valida los datos y los certificados.
   - Si la validación es satisfactoria, el *PrivacyCA* genera un certificado AIK y lo cifra con clave simétrica.
   - Este blob simétrico también contiene el número aleatorio R1 que ha recibido antes y el nuevo número aleatorio generado por el *PrivacyCA* (R2).
   - La clave simétrica y el resumen (*hash*) de la clave pública IAK es cifrado con la clave pública EK del TPM, obtenido del certificado EK recibido en la solicitud anterior. Esta estructura es el blob asimétrico.
   - El paquete que se enviará esta formado por el blob simétrico y el asimétrico asegurando que esta respuesta sólo la pueda descifrar el TPM al que va dirigido.
   - Después de construir la respuesta, esta es enviada al cliente que generó la solicitud.

3. El cliente comprueba los datos recibidos del *PrivacyCA:*

   - El cliente hace una llamada al método receiveResponse que invoca a su vez a la función TSS Tspi_TPM_ActivateIdentity donde se descifra la respuesta recibida.
   - Posteriormente el TSS descifra la respuesta a través del módulo TPM. Primero se descifra el blob asimétrico con la clave EK privada. Si el AIK que contiene dicho blob está disponible en el TPM, la clave simétrica es devuelta y ésta se utiliza para descifrar el blob simétrico.
   - El cliente comprueba si el número aleatorio recibido del *PrivacyCA* es igual que el que había generado él anteriormente. Finalmente, el cliente envía el número aleatorio R2 al *PrivacyCA*, en este caso va en claro.

4.3 Autenticación segura sin la entidad PrivacyCA

Este tipo de autenticación comienza con un intercambio de claves inicial, la clave pública AIK y la clave pública EK. Los diferentes nodos deben tener su módulo TPM activado ya que, en este caso, el proceso de autenticación cifrará y descifrará con cada TPM. Cada máquina debe ser capaz de cifrar los datos de forma que sólo sea capaz de descifrar los mismos la máquina destino. Esto se ha hecho con el intercambio de claves AIK públicas. Sólo si esta habilitada en el TPM la clave AIK será valida. Además, las claves públicas EK son intercambiadas con la intención de que cada TPM cifre los datos utilizando la clave pública del otro TPM.

El protocolo de autenticación segura sin la entidad *PrivacyCA* está ilustrado en la siguiente figura y se describe a continuación.

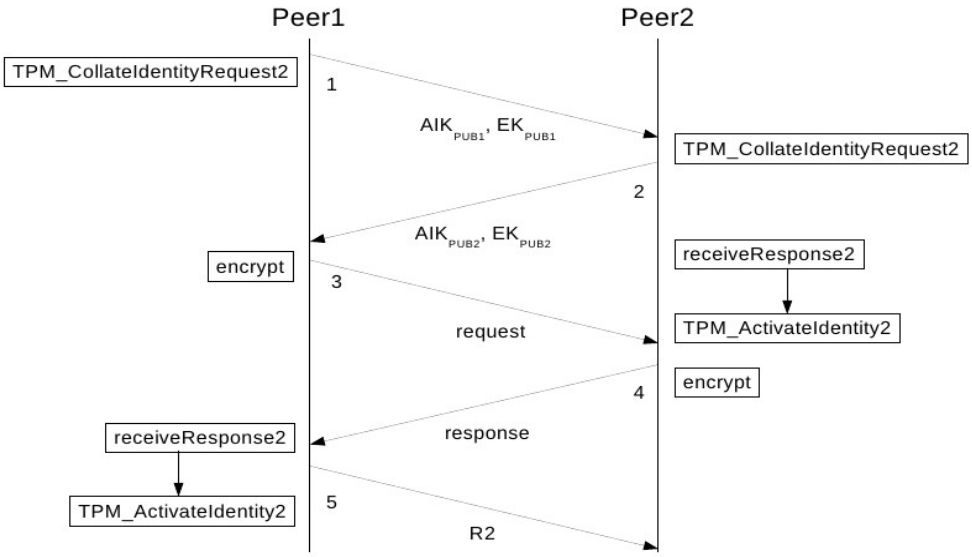


Figura 3: Trusted Authentication Protocol sin PrivacyCA

1. El nodo 1 invoca a la función TPM_CollateIdentityRequest2 para generar el par de claves llamado AIK *(Attestation Identity Key)*. Este envía al otro nodo la clave pública EK y la clave pública AIK.

2. El nodo 2 invoca a la función TPM_CollateIdentityRequest2 para generar el par de claves llamado AIK (Attestation Identity Key). Este envía al otro nodo la clave pública EK y la clave pública AIK.

3. El nodo 1 genera un número aleatorio (R1) y una clave simétrica. La clave simétrica con un resumen de la clave pública AIK del nodo 2 es cifrado con la clave pública EK del nodo 2, estas claves son las obtenidas en la fase inicial de intercambio de claves. Esta estructura se llama blob asimétrico. El paquete formado por el blob simétrico y el asimétrico asegura que esta solicitud solo puede ser descifrada por el TPM al que se lo envía.

4. El nodo 2 invoca a la función receiveResponse2 para descifrar los datos haciendo una llamada a la función TPM_ActivateIdentity2. De esta forma se obtiene el número aleatorio R1. Posteriormente, el nodo 2 genera un número aleatorio R2 y lo cifra con R1 siguiendo el mismo proceso que hizo el nodo 1 en el punto anterior.

5. El nodo 1 recibe R1 y R2 descifrando del mismo modo que el nodo 2 hizo. Una vez consigue estos dos números comprueba que el número R1 que acaba de recibir es igual al que él generó anteriormente. Finalmente, el nodo 1 envía al nodo 2 el número R2 pero esta vez sin cifrar, el nodo 2 al recibirlo comprobará que es el mismo número que él mismo generó antes.

5 Bibliografía

[1] M. Pirker, R. Toegl, T. Winkler and T. Vejda. Trusted Computing for the JavaTM platform.
http://trustedjava.sourceforge.net/

[2] Trusted Computing Group.
http://www.trustedcomputinggroup.org/.

[3] A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography, CRC Press, 1996. Chapter 10: Identification and Entity Authentication.
http://www.cacr.math.uwaterloo.ca/hac/.

[4] Stefan Kraxberger, Martin Pirker and Ronald Tgl. Trusted Identity Management for P2P Networks. 2011.

[5] S. Balfe, A.D. Lakhani and K.G. Paterson. Securing Peer-to-Peer networks using Trusted Computing. In C.J. Mitchell (ed.), Trusted Computing, IEE Press, 2005, pp.271-298.

[6] Martin Pirker, Ronald Toegl, Daniel M. Hein, Peter Danner. A PrivacyCA for Anonymity and Trust. TRUST 2009, pp.101-119.

[7] Stefan Kraxberger. Sepp Framework.

[8] Trusted Computing Group, “TPM main specification”,
http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[9] Trusted Computing Group, “TCG Software Stack (TSS) Specification”,
http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification.

[10] Trusted Computing Group, “TCG architecture overview 1.4”,
http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14.

[11] Infineon TPM 1.2 Specifications,
http://www.infineon.com

[12] TPM Emulator from ETH Zurich (Software).

[13] IAIK jTSS - TCG Software Stack for the Java (tm) Platform
http://trustedjava.sourceforge.net/index.php?item=jtss/readme

[14] IAIK jTpmToolsTPM Tools for the Java (tm) Platform
http://trustedjava.sourceforge.net/index.php?item=jtt/readme

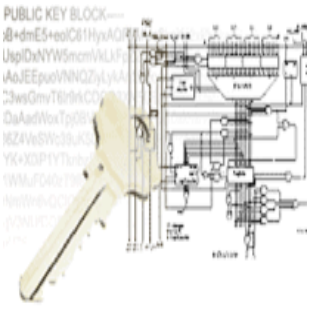
[15] IAIK/OpenTC TCcert - Trusted Computing certificate tool
http://trustedjava.sourceforge.net/index.php?item=tccert/readme

[16] IAIK/OpenTC PrivacyCA documentation
http://trustedjava.sourceforge.net/index.php?item=pca/apki

6 Anexo: Application of Trusted Computing to secure P2P networking

Graz University of Technology
IAIK
Institute for Applied Information Processing and Communications
Inffeldgasse 16a
A–8010 GRAZ

Bachelor Thesis



Application of Trusted Computing to secure P2P networking

Version 0.1

Elisa Pintado Guijarro 1030152, Guillermo Garcia Millan 1030149

Article

Bachelor Thesis - T 705.107

http://www.iaik.tugraz.at/content/teaching/

Application of Trusted Computing to secure P2P networking

Elisa Pintado Guijarro, Guillermo Garcia Millan

July 8, 2011

1 Introduction

Peer-to-peer (P2P) networking is based on a distributed application architecture that covers a diverse set of network types. In pure P2P overlays, every node in the network acts as a *servent*, they act as a server and a client simultaneously. Highlighting the fact that there is a lack of any centralised control, acting all nodes as equals.

This networks have become very popular in the form of file-sharing networks. It is based on the idea of sharing any type of resources between all nodes. Peers are both suppliers and consumers of resources. As nodes arrive, the total capacity of the system also increases. In contrast, in a traditional client-server architecture, clients share only their demands with the system, but not their resources. In this case, as more clients join the system, fewer resources are available to serve each client. The decentralized nature of P2P networks also increases robustness because if a part of the system fails, it will not stop the entire system from working.

As mentioned in[5] there is a need to provide robust access control, data integrity, confidentiality and accountability services. In order to prevent other nodes from impersonating or creating an arbitrary amount of bogus nodes, all distributed systems must have a unique, undeniable and verifiable identifier for each node. The foundation of stable and verifiable identities is required to build nodes with secure parameters. The use of unsecure nodes may allow remote access to files on a victim's computer or even compromise the entire network, this is explained in [4]. Finding suitable measures against the increasing variety of software-based attacks is a difficult task. In particular, pseudospoofing attacks, in which malicious parties claim multiple identities and disrupt the operation of P2P networks.

The Trusted Computing paradigm offers a very useful and powerful set of security features to improve on computer systems. The Trusted Computing Group (TCG)[2] specifies the Trusted Platform Module (TPM), wich allows to provide cryptographically qualified and tamper-resilient statements on the software configuration of a machine. As described in [6], the use of protected cryptographic mechanisms alone is not sufficient to convince remote machines or human users that a complex software service can actually be trusted. To enable a decision based on the statements made by a TPM and the associated trust

levels, keys need to be vouched for. This requires a Public Key Infrastucture (PKI). Allowing a remote and indipendant party to decide on the trustworthiness of a host or a particular service. The TPM provides authenticity and allows unique identification of a platform, therefore creating a privacy problem. To circumvent this problem the TCG proposed a trusted third party, the Privacy Certification Authority (PrivacyCa). A particular incarnation of the PKI concept is the PrivacyCa, a trusted service capable of reporting its state to clients. It confirms that keys are protected by a specification-compliant TPM implementation and thus may be trusted under certain conditions, but without revealing the specific identity of the TPM and the user.

In this work we provide a solution for creating unique, undeniable and verifiable identifiers for P2P networks by using the security features that the TPM offers. We implement a protocol called Trusted Authentication protocol (TAP) offering two possible solutions for authentication. One based on machines with TPM hardware enabled and a PrivacyCa that dictates if a machine can be trusted. The other solution is based on the authentication of machines using their own TPM without a PrivacyCa. The main goals of a secure P2P would be to identify and avoid malicious nodes, this is achieved by secure authentication of machines via TPM.

2 Background

The issues that exist on P2P networks are a well known security threat. There has been many researching in order to identify and try to solve some of this issues.

Theoretically, all nodes that belong to a P2P network will provide equal amounts of resources to the system. But this is not always true, some nodes try to disturb the execution of the system. Depending on the P2P network that we have several problems may arise. For example, there are nodes that try to infest a network with malicious files, others try to consume many bandwith and share few resources with the rest of nodes. As described in [4], the problems in P2P overlays can be separated into two domains: system instrinsic and application specific.

Stefan Kraxberger et al. [4] provides a solution based in the use of the information and keying material enclosed in the Trusted Computing's TPM. This TPM is conceived as Root-of-Trust for storage and attestation.

In our work we describe different approaches, in order to solve issues regarded to authentication in P2P networks. Our solution is based on the guidelines provided by the IAIK group in [4] where a thorough explanation about Trusted Identity management for P2P networks is described.

2.1 Trusted Computing Platforms

The Trusted Computing Group (TCG) [2] is an international industry standards group. This group has defined a set of specifications to evolve current computer architectures into Trusted Platforms. To provide a hardware anchor for trust, the Trusted Computing Group has specified the Trusted Platform Module (TPM)[8]. The TPM contains hardware implementations of cryptographic operations for public key cryptography, key generation, hashing and random number generation. This hardware has capabilities beyond traditional tokens or smart cards, ensuring only authorized users and authorized machines to be on the network.

The TPM provides an encryption key that is permanently embedded inside the hardware called the Endorsement Key (EK). This is done generally at the time of manufacture. The private part of the EK is never released outside of the TPM, this way the public part of the EK is used to recognize the genuine TPM. A practical feature that makes use of the EK involves signing pieces of data to allow other components to verify that the data can be trusted. To sign a piece of data, a private key is used to encrypt a small piece of information. The signature can be verified by using the corresponding public key to decrypt that same piece of data. If it can be decrypted with the public key, then it must have been encrypted by the corresponding private key. As long as that private key has been kept secret, this digital signature can be trusted.

The TCG provides a software infrastructure called the TCG Software Stack (TSS)[9]. Applications can access Trusted Computing functionality by using the Trusted Service Provider (TSP) interface. The Trusted Core Services (TCS) are implemented as a single system service. The TCS communicates with the TPM via the TSS Device Driver Library (TDDL).

To prevent tampering with the platform measurements the TPM stores them in a set of Platform Configuration Registers (PCRs). A report on the PCR state reflects the exact state of a system, its authenticity and integrity must be preserved. With this report an external stakeholder can form an informed opinion about a system's trustworthiness. This central concept of a TC-enabled platform is known as Remote Attestation. To this end, the TPM acts as Core Root-Of-Trust for Reporting (CRTR). Upon request, it performs the TPM_Quote operation which signs the PCR state with a TPM hosted signing-capable key. In remote attestation, it is important to know that you are communicating with a valid TPM-enabled platform. EK can be used, but its uniqueness would undermine the privacy of the owner. For this a specific key is created, called Attestation Identity Key (AIK). To ensure that the signature on an attestation report is actually made by a TPM-protected AIK, the used key must be vouched for within the framework of a PKI.

2.2 PrivacyCA

In TCG remote attestation it is required for one machine to send a detail description of its system state, this can be highly susceptible to attacks. A logical solution is using a PrivacyCA as a Trusted Entity, protecting the privacy of the users. More information about the PrivacyCA is detailed in [6]. One objective of attestation is to allow the Challenger to determine that some TPM has signed a message, without knowing which TPM signed the message.The TCG specifications [10] define the PrivacyCA service to be employed to issue AIK credentials that vouch for the trustworthiness of a platform without disclosing EK unique values to a Challenger. The TPM enrolls AIK public keys with a PrivacyCA. The PrivacyCA may then distribute a credential certifying the AIK. Enrollment with a PrivacyCA requires the TPM to prove AIK keys are exclusively bound to the TPM. The platform accomplishes this by decrypting the AIK credential using the EK private key in the TPM. Only the TPM with the EK private key will be able to perform the decryption. The Privacy CA is trusted not to misrepresent the trust properties of platforms for which AIK credentials are issued.

2.3 Sepp Framework

The Secure P2P framework [7] is a project that develops a secure P2P framework in Java (SePP). Existing P2P frameworks are not covering security by design but rather implement only some high layer security mechanisms. In our work we implement new functionalities based on authentication with trusted computing. This framework is used as a tool for obtaining a secure P2P network where we can develop our solution and measure times for different authentications.

3 Configuration of the Trusted Platform Module

In our application, we assume that machines are TPM enabled. This section tries to give some guidelines on how to achieve the correct execution of the TPM hardware. For this, we show the hardware specification used in this work and which Java Packages were useful.

3.1 System requirements

The TPM hardware employed is Intel Infineon Trusted Platform Module, version 1.2 [11]. Enabling the TPM in the BIOS is necessary. If your machine does not have a TPM hardware, it is possible to obtain a TPM emulator [12]. The TPM emulator is a software package for Linux operating systems providing TPM functionality as a pure software implementation.

The operating system used in our work is Windows 7, the objective in our work was to develop our platform giving support to this operating system, offering an alternative to Linux.

For implementation, Sun Java (tm) Environment of version 5 or later is used. Earlier Java (tm) versions do not provide the required cryptographic functionality. Compatibility with other Java vendors is unknown and untested. The framework used has been Eclipse. Our software has been implemented using already developed frameworks, the packages needed are the following.

- jTSS
- jUnit
- jTPMTools
- TCcert Tool
- PrivacyCA

3.2 Enabling the Trusted Platform Module

If you use a hardware TPM you first have to activate it in your BIOS. For more information, look into the manual of your computer. A manual setup of JTSS for linux and windows is described in [13]. Also, the ownership of the TPM has to be taken, when this is done, the storage root key (SRK) will be stored in the system storage (without the private key part). For taking ownership, we use jTpm Tools, there is a detailed explanation of how to do this in [14]. If the ownership has been already taken you can use the clear_owner command, this way all TPM protected data will be lost. To take ownership of a TPM, take_owner command is used. Notice that this should be done under administrator privileges under Windows.

A TPM is initialized with an owner password, a srk password, an aik password and an aik id label. In Java this is done in the following way.

- owner password
  ownerSecret = TcBlobData.newString("mypassword", appendNullTerm, encoding);
- aik password (required)
  aikSecret = TcBlobData.newString("mypassword", appendNullTerm, encoding);
- srk password
  srkSecret = TcBlobData.newByteArray(TcTssConstants.TSS_WELL_KNOWN
  _SECRET);
  srkSecretMode = TcTssConstants.TSS_SECRET_MODE_SHA1;
- aik id label (required)
  idLabel = TcBlobData.newString("myidlabel", false);

3.3 Other tools

IAIK/OpenTC TCcert [15] is a software tool which enables one to create special types of certificates, as specified by the Trusted Computing Group. IAIK/OpenTC PrivacyCA [16] is a package which offers a basic implementation of a PrivacyCA server.

4 Implementation

Our application is implemented based on [1] where we have modified existing structures and created new methods in order to achieve our approach. This solution has been integrated in the Sepp Framework [7], where we can test our code in a real P2P Network using as example a client based chat.

The Sepp Framework has been modified and we have added new functionality. These features include two different types of authentication that the framework did not provide:

- Trusted Authentication with Privacy CA.
- Trusted Authentication without Privacy CA.

The first one uses an entity Privacy CA as a Trusted Third Party (TTP). The second approach allows Trusted Authentication without this Privacy CA entity using only the TPM machines, which are used as Root-of-Trust.

4.1 Trusted Authentication Protocol

The Trusted Authentication Protocol (TAP) is used in both new authentication types. TAP is based on Needham-Schroeder PK Protocol. This protocol adds random numbers in the authentication process as explained in chapter 10 of [3].

Figure 1 shows the exchange of messages that take place in the authentication process, for both, Trusted Authentication with Privacy CA and without it.

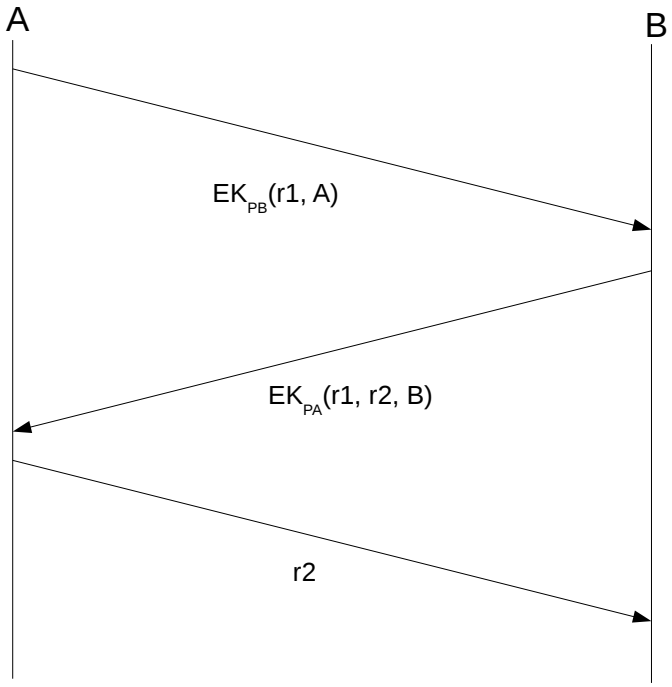


Figure 1: Trusted Authentication Protocol (TAP)

A Node A

B Node B

EK_{PB} Endorsement Public Key of node B

EK_{PA} Endorsement Public Key of node A

R1 Random number 1

R2 Random number 2

In the first sending, node A sends to node B a message with its identification and a random number R1 generated by this node, data is encrypted with B's public key. Further, B receives the encrypted data blob and decrypts it with its private part, this way it obtains R1. In the next step, B sends to A its identification with the previous random number R1 and generates a random number R2, all this data is encrypted in this case with the public key of A. Finally, A receives in the same way an encrypted data blob with R2 and it decrypts the number to send it back to the node B, this message is not encrypted. If the

number that a node generates is equal to the number that it gets back, then the process will succeed.

The following sections explain in more detail how TAP is implemented with both authentication types.

4.2 Trusted Authentication with Privacy CA

This approach is based on the solution with Trusted Identity Management for P2P Networks [4], where it is explained how to avoid the current problems with authentication in P2P networks.

We have implemented a new class called TPMmanager that handles communication with the TPM hardware. In order to complete our solution, we need to modify several libraries from the trusted java framework [1] to use them in our program, these changes will be detailed further on.

The process starts when a user wants to connect to a network, this user will be called attestant. This attestant has an associated AIK certificate that does not contain any information that could link to the specific platform hosting the AIK. To avoid the creation of possible identity attacks, a PrivacyCA has to check that a certain TPM or endorsement key certificate can only issue one AIK certificate.

The flow that we follow is based on the AIK protocol described in [4].

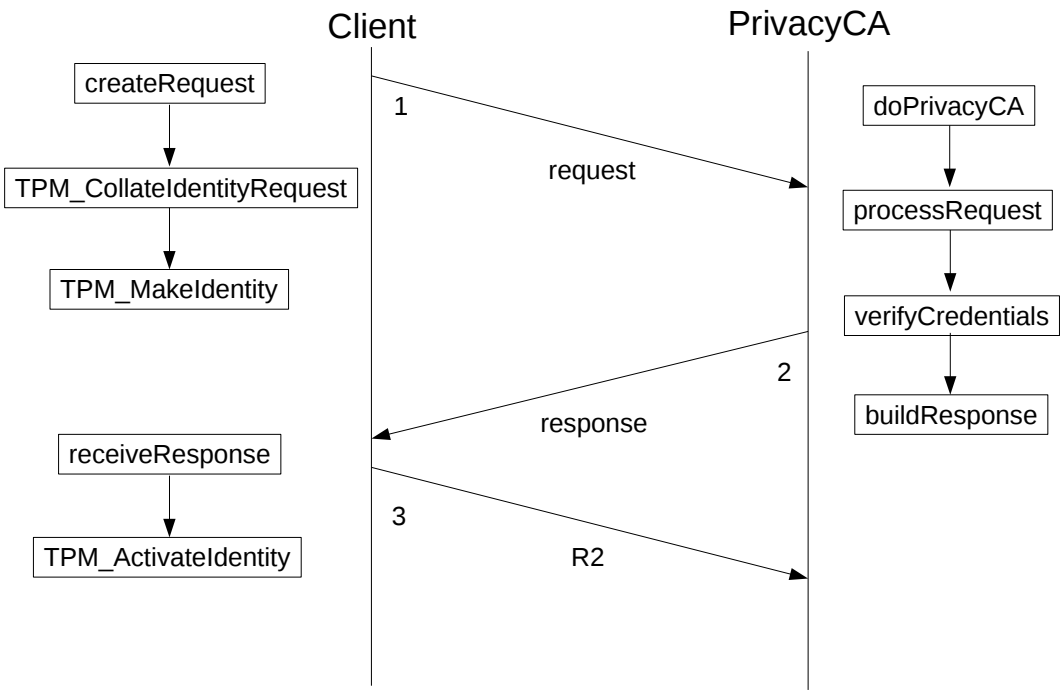


Figure 2: Trusted Authentication Protocol with PrivacyCA

1. The client prepares a request to send to the PrivacyCA :
   - In the first step the client calls the method createRequest that invokes the Tspi_TPM_CollateIdentityRequest TSS function to initiate a new AIK creation.
   - The TSS invokes the TPM_MakeIdentity TPM function to create the new attestation identity key (AIK) that is a RSA key pair.
   - The TPM returns a structure containing the public AIK, signed with the private AIK key. And then, the TSS generates a request where the data is stored in an identityProof structure containing the signed blog returned by the TPM, the first random number R1 and the EK and PE certificates of the platform.
   - This data is encrypted with the public key of the PrivacyCA.
   - The client sends to the PrivacyCA this request.
2. The PrivacyCA prepares a response to send to the client :
   - The PrivacyCA calls the method processRequest where it decrypts the data, and then it validates its content and the certificates contained in the request.

- On successful validation the PrivacyCA issues an AIK certificate, encrypted with a symmetric key.
- This symmetric blob also contains the random number R1 that was received before and a new random number generated by the PrivacyCA (R2).
- The symmetric key along with a hash of the public AIK is encrypted with the public key of the EK of the TPM, obtained from the EK certificate of the request. This structure is called asymmetric blob.
- The result package formed by the symmetric and asymmetric blobs assure that the response can only be decrypted by the intended recipient TPM.
- After building the response, it is sent back to the client.

3. The client checks the received data from the PrivacyCA:

   - Then, the client calls the receiveResponse method that invokes the Tspi_TPM_ActivateIdentity TSS function where it decrypts the response.
   - The TSS subsequently requests the platforms TPM to decrypt the response package with the private part of the EK. If the referenced AIK is available on the TPM, the symmetric key is returned and it is used to decrypt the symmetric blob contained in the response.
   - The client checks if the random number received from the PrivacyCA is the same number that was generated. Finally, the client sends R2 to the PrivacyCA, in this case without encryption.

4.3 Trusted Authentication without PrivacyCA

This authentication type begins with an initial exchange of keys, the AIK public key and the Endorsement public key. Having different peers with their TPM enabled, the authentication process will make encryptions and decryptions with each TPM. Each machine must be able to encrypt the data in a way that only the target machine is able to decrypt this information. This is done with the exchange of the AIK public keys. If previously the TPM has enabled this AIK key then it will be a valid key. Also, the Endorsement public key is exchanged with the intention for each TPM to be able to encrypt data using the public key of the other TPM.

The Trusted Authentication Protocol without PrivacyCA is illustrated in Figure 3 3 and outlined below.

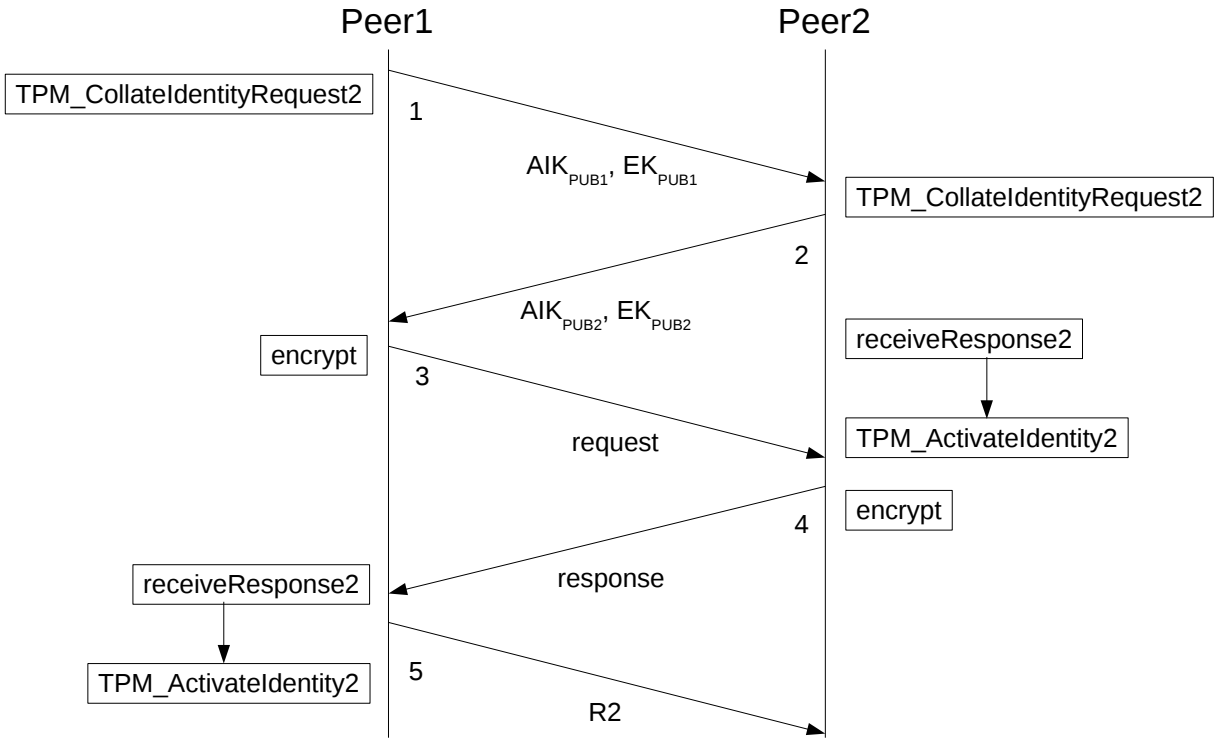


Figure 3: Trusted Authentication Protocol without PrivacyCA

1. Peer 1 invokes TMP_CollateIdentityRequest2 to generate a key pair called Attestation Identity Key (AIK). It exchanges the Endorsement Public Key (EK_{PUB1}) and the Attestation Public Identity Key (AIK_{PUB1}).

2. Peer 2 invokes TMP_CollateIdentityRequest2 to generate a key pair called Attestation Identity Key (AIK). It exchanges the Endorsement Public Key (EK_{PUB2}) and the Attestation Public Identity Key (AIK_{PUB2}).

3. Peer 1 generates a random number (R1) and a symmetric key. The symmetric key with a hash of AIK_{PUB2} is encrypted with EK_{PUB2}, obtained in the initial phase (2). This structure is called asymmetric blob. R1 among other data is encrypted with the symmetric key creating a symmetric blob. The result package formed by the symmetric and asymmetric blobs assure that the request can only be decrypted by the intended recipient TPM.

4. Peer 2 invokes receiveResponse2 to decrypt the data calling TMP_ActivateIdentity2. This way we obtain R1. Afterwards, peer 2 generates a random number R2 and encrypts R1 and R2 following the same process as peer 1.

5. Peer 1 receives R1 and R2 decrypting in the same way as peer 2 did. Then it checks that the received R1 corresponds with the R1 that was sent before. Finally, peer 1 sends R2 without any encryption to peer 2, who will check if R2 matches.

5 TPM Time Measures

The implemented application accesses the TPM hardware of the machine in multiple times to make different operations such as credential creation, data encryption and decryption.

With a PrivacyCA, it has to build a new AIK credential using the arguments received by the TPM client.

This section analizes these two attestation types and in it, we measure the time invested in the attestation process. How long it takes to execute the main operations from the TPM and the PrivacyCA, such as encrypt and decrypt.

These results will show which options could be better to use and the possible benefits and disadvantages. The best option would be to prescind from the intermediate PrivacyCA, which spends more operational time building new credentials and does not agree with the P2P philosophy, in which all nodes are equal and there is no central service with more privileges. In the other hand, the solution without the presence of this entity, using only machines provided with TPM and making the attestation process between them, is faster as shown in next subsections.

5.1 Trusted Authentication with PrivacyCA

The following section shows the resulting tests done for the Trusted Athentication Process with PrivacyCA. We have made 3 consecutive tests to measure times between the different parts of execution.

createRequest It implements the creation of AIK credentials and builds an encrypted request using the PrivacyCa public key.

doPrivacyCA It decrypts the request received from the client, it extracts the first random number (R1) and also the certificates of the client. After this is done, it generates a new random number (R2), and then the PrivayCA creates a response that is composed of two parts. The symmetric part, that contains a PrivacyCA credential, R1 and R2; and the asymmetric part, that contains among other data the symmetric key, which is used to decrypt the symmetric part. The asymmetric part is encrypted with the endorsement public key of the client and using the AIK public key to assure that only the intended client will be able to decrypt the response.

receiveResponse It receives the response from the PCA and decrypts it using the private key that is stored inside the TPM.

The results of this process is shown in the following table, the last row shows the total time of execution that the process takes.

Table 1: Test Time Measures for TAP with PCA

Test	1	2	3
createRequest	00:03:50	00:03:46	00:03:53
doPrivacyCA	00:21:15	00:18:82	00:19:80
receiveResponse	00:02:34	00:02:36	00:02:50
Total	00:26:85	00:24:64	00:25:82

The following figure 4 illustrates the proportion of time dedicated to each phase.

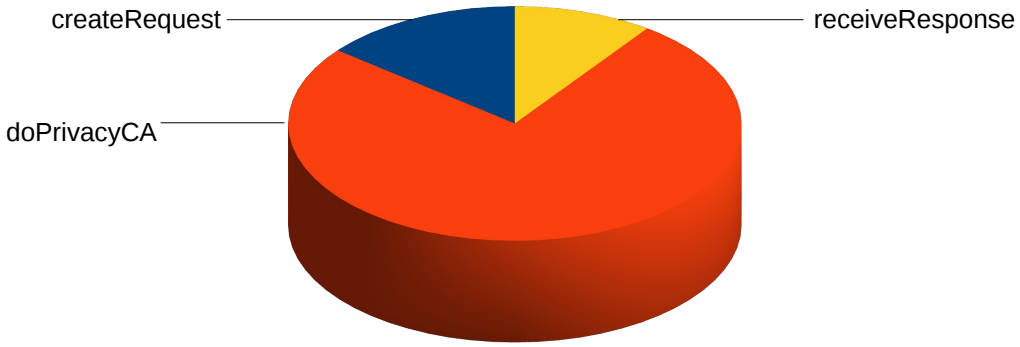


Figure 4: Time Measures for TAP with PCA

The PrivacyCA wastes a lot of time in preparing the encryption to allow a correct decryption in the TPM. This is due to this process has to create a new AIK credential based in the credentials received from the TPM in order to build a well formed response.

5.2 Trusted Authentication without PrivacyCA

The following section shows the resulting tests done for the Trusted Athentication Process without PrivacyCA. We have made 3 consecutive tests to measure times between the different parts of execution.

initial phase This phase calls initialy the createRequest2 method, which creates the AIK credential using the TPM. In this phase we also obtain the Endorsement public key of the TPM. When this is done, the AIK public key and the Endorsement public key is exchanged between machines.

encrypt It generates the random number R1, and prepares the encryption with AIK public key and the Endorsement public key of the other TPM. This way the intended TPM will be able to decrypt the data.

receiveResponse2 It decrypts with its TPM the encryption blob received from the other machine. This encryption blob consists in a symmetric and asymmetric blob structure as seen in authentication with PrivacyCA.

The results of this process is shown in the following table, the last row shows the total time of execution that the process takes.

Table 2: Test Time Measures for TAP without PCA

Test	1	2	3
initial phase	00:03:39	00:03:42	00:03:34
encrypt	00:00:17	00:00:17	00:00:18
receiveResponse2	00:02:46	00:02:39	00:02:42
Total	00:05:52	00:05:49	00:05:78

The following figure 5 illustrates the proportion of time dedicated to each phase.

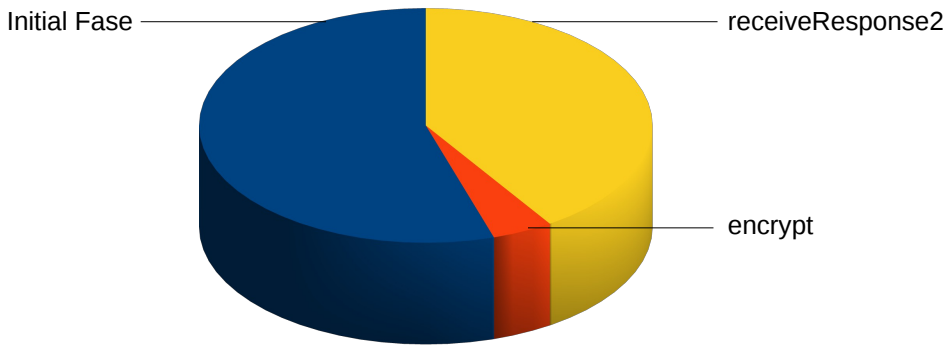


Figure 5: Time Measures for TAP without PCA

With this authentication process and without using a PrivacyCA, as it can be observed we earn an average of 20 seconds. With these results we can confirm that this authentication type for P2P networks is faster than using authentication with PrivacyCA.

6 Related Work

Trusted computing in machines is practically a new born technology that still can be improved in different ways. The possibility of not just to secure the hardware for its owner, but also to secure it against its owner is controversial. A TPM equipped computer is able to uniquely attest to its own identity, using the attestation key will make possible for vendors to identify the user. To avoid this loss of anonymity we use a PrivacyCA implementation and also we develop a way of authentication without this third trusted party concept based on Direct Anonymous Attestation (DAA) as proposed in TPM v.1.2. Anyway this service infrastructure was so far a theoretical concept and it can be used as a basis that can evolve in new research developments.

7 Conclusions

In this work we investigate different possibilities that are presented using the Trusted Platform Module to enable the provision of unique trustworthy identities. One possibility is to use a PrivacyCa as a central node trusted by the rest of the nodes in the network. The other possibility explored on this work is to create an special trusted P2P network, where any existing node inside the network can authorise the integration of new nodes in the system.

Anyway, both possible authentications show their own benefits and disadvantages. The best advantage that exists without using an intermediate PrivacyCA leans on having a network where all nodes are equal and there is no central service with more privileges. Furthermore, as we have seen measuring times the PrivacyCA spends more operational time building new credentials. Finally, to have a PrivacyCA will assure in a more robust manner the anonymity of a node. The possibility of creating new credentials using a trusted central service makes much more difficult to trace the origin of a certain node. As we can state, using trusted computing with secure P2P via TPM authentication offers a great advantage, identifying and avoiding malicious nodes. This way we can have more control securing P2P networks.

References

[1] M. Pirker, R. Toegl, T. Winkler and T. Vejda. Trusted Computing for the Java^{TM}platform.
http://trustedjava.sourceforge.net/, 2009.

[2] Trusted Computing Group.
http://www.trustedcomputinggroup.org/.

[3] A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography, CRC Press, 1996. Chapter 10: Identification and Entity Authentication.
http://www.cacr.math.uwaterloo.ca/hac/.

[4] Stefan Kraxberger, Martin Pirker and Ronald Tgl. Trusted Identity Management for P2P Networks. 2011.

[5] S. Balfe, A.D. Lakhani and K.G. Paterson. Securing Peer-to-Peer networks using Trusted Computing. In C.J. Mitchell (ed.), Trusted Computing, IEE Press, 2005, pp.271-298.

[6] Martin Pirker, Ronald Toegl, Daniel M. Hein, Peter Danner. A PrivacyCA for Anonymity and Trust. TRUST 2009, pp.101-119.

[7] Stefan Kraxberger. Sepp Framework.

[8] Trusted Computing Group, "TPM main specification",
http://www.trustedcomputinggroup.org/resources/tpm_main_specification.

[9] Trusted Computing Group, "TCG Software Stack (TSS) Specification",
http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification.

[10] Trusted Computing Group, "TCG architecture overview 1.4",
http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14.

[11] Infineon TPM 1.2 Specifications,
http://www.infineon.com.

[12] TPM Emulator from ETH Zurich (Software).

[13] IAIK jTSS - TCG Software Stack for the Java (tm) Platform
http://trustedjava.sourceforge.net/index.php?item=jtss/readme

[14] IAIK jTpmToolsTPM Tools for the Java (tm) Platform
http://trustedjava.sourceforge.net/index.php?item=jtt/readme

[15] IAIK/OpenTC TCcert - Trusted Computing certificate tool
http://trustedjava.sourceforge.net/index.php?item=tccert/readme

[16] IAIK/OpenTC PrivacyCA documentation
http://trustedjava.sourceforge.net/index.php?item=pca/apki